RECONSTRUCTING MEMORY RESIDENT QUEUES OF AN INACTIVE PROCESSOR

Cross-Reference to Related Applications

[0001] This application contains subject matter which is related to the subject matter of the following applications, which are assigned to the same assignee as this application. The below listed applications are hereby incorporated herein by reference in their entirety:

[0002] "HIGH-PERFORMANCE MEMORY QUEUE", Chen et al., (IBM Docket No. POU920000185US1), Serial No. 09/790,853, filed February 22, 2001;

[0003] "PERFORMANCE OF CHANNELS USED IN COMMUNICATING BETWEEN SENDERS AND RECEIVERS", Chen et al., (IBM Docket No. POU920000186US1), Serial No. 09/790,781, filed on February 22, 2001; and

[0004] "MANAGING MEMORY RESIDENT QUEUES TO CONTROL RESOURCES OF THE SYSTEMS USING THE QUEUES", Chen et al., (IBM Docket No. POU920000198US1), Serial No. _____, filed _____.

Technical Field

[0005] This invention relates, in general, to network messaging and queuing, and in particular, to taking over

and/or reconstructing memory resident queues of an inactive processor.

Background of the Invention

[0006] One technology that supports messaging and queuing across a network is referred to as MQSeries and is offered by International Business Machines Corporation, Armonk, New York. With MQSeries, users can dramatically reduce application development time by using MQSeries API functions. Since MQSeries supports many platforms, MQSeries applications can be ported easily from one platform to another. In a network, two MQSeries systems communicate with each other via channels, such as MQSeries channels. An MQSeries sender channel defines a connection from one MQSeries system to another MQSeries system and transmits messages from the one system to the other system.

[0007] To facilitate transmission of messages from one system to another system, memory resident queues are used. In particular, messages are written to and retrieved from the queues. Messages put to a queue are guaranteed to be delivered to their final destination. That is, messages have persistence. It is also guaranteed that only a single copy of the message will be delivered. Also, messages are delivered in a time-independent, asynchronous manner. This is beneficial, since applications may continue processing regardless of the status of the underlying network.

[0008] In order to maintain persistence for MQSeries messages, the most common implementation for MQSeries hosts is to have each queue reside on direct access storage device (DASD) media. These queues are referred to as DASD resident queues. DASD resident queues provide a benefit of being easily movable between processors in a parallel complex. This is because they contain the complete up-to-date version of the queues to be moved with no intermediate steps necessary. However, DASD resident queues have proven inadequate in terms of performance. Thus, high performance, high access queues now reside in memory, and are referred to as memory resident queues.

[0009] Although memory resident queues have obvious performance improvements over DASD resident queues, difficulties arise when a processor housing a memory resident queue becomes inactive. In this situation, the memory resident queue becomes inaccessible to other processors that may need the queue, until the inactive processor is brought back online. Thus, any messages stored on those queues cannot be delivered to their final destination in a timely manner.

[0010] Based on the foregoing, a need exists for a capability that enables an active processor to take over a queue for an inactive processor. A further need exists for a capability that enables an active processor to reconstruct a memory resident queue for an inactive processor.

Summary of the Invention

[0011] The shortcomings of the prior art are overcome and additional advantages are provided through the provision of a method of switching queue ownership. The method includes, for instance, obtaining an indication that a queue is to be taken over, the queue being resident in memory of a first processor; and moving the queue from the first processor to a second processor, the queue to be resident in memory of the second processor.

[0012] In a further embodiment, a method of reconstructing queues is provided. The method includes, for instance, rebuilding contents of a queue to obtain an updated version of the queue, the queue being a memory resident queue of a first processor; and reading at least a portion of the updated version of the queue into memory of a second processor, the second processor being different than the first processor.

[0013] System and computer program products corresponding to the above-summarized methods are also described and claimed herein.

[0014] Advantageously, a capability is provided that enables an active processor to rebuild and take over one or more memory resident queues of an inactive processor. The ability to move ownership of a queue (or messages on the queue) from an inactive processor to an active processor enables those messages to be processed by an application or

sent on to the next destination without waiting for the inactive processor to be brought back into the complex.

[0015] Additional features and advantages are realized through the techniques of the present invention. Other embodiments and aspects of the invention are described in detail herein and are considered a part of the claimed invention.

Brief Description of the Drawings

[0016] The subject matter which is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the invention are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

[0017] FIG. 1a depicts one embodiment of a communications environment incorporating and using one or more aspects of the present invention;

[0018] FIG. 1b depicts one example of various components of an operating system of FIG. 1a, in accordance with an aspect of the present invention;

[0019] FIG. 2 depicts one embodiment of a local queue and associated checkpoint and recovery log,

in accordance with an aspect of the present invention;

[0020] FIG. 3 is a pictorial illustration of an inactive processor, which has a queue with two messages, one in its checkpoint and one in its recovery log;

[0021] FIG. 4 is a pictorial illustration of an active processor rebuilding the queue of the inactive processor of FIG. 3, in accordance with an aspect of the present invention;

[0022] FIG. 5 is a pictorial illustration of the active processor replacing the checkpointed version of the queue with a complete rebuilt version of the queue, in accordance with an aspect of the present invention;

[0023] FIG. 6 depicts one embodiment of a general overview of the logic associated with reconstructing a queue, in accordance with an aspect of the present invention;

[0024] FIGs. 7a-7b depict one embodiment of further details of the logic associated with rebuilding the queue of FIG. 6, in accordance with an aspect of the present invention;

[0025] FIGs. 8a-8b depict one embodiment of further details of the logic associated with switching ownership of the queue of FIG. 6, in accordance with an aspect of the present invention; and

[0026] FIG. 9 depicts one embodiment of further details of the logic associated with reflecting the ownership change of the queue of FIG. 6, in accordance with an aspect of the present invention.

Best Mode for Carrying Out the Invention

[0027] In accordance with an aspect of the present invention, a capability is provided which enables one processor (e.g., an active processor) of a communications environment to take over one or more queues of another processor (e.g., an inactive processor) of the environment. In a further aspect of the invention, at least one processor of the environment reconstructs one or more memory resident queues of another processor of the environment.

[0028] As one example, a communications environment incorporating and using one or more aspects of the present invention is a scalable parallel complex having a plurality of computing units. Each computing unit includes one or more processors, and each processor may be a sender of messages, a receiver of messages, or both. One embodiment of a sender and a receiver of messages is described with reference to FIG. 1a.

[0029] As shown in FIG. 1a, in a communications environment 100, a sender 102 is coupled to a receiver 104 via one or more channels 106. In one example, sender 102 includes an operating system 108, such as the TPF Operating System offered by International Business Machines Corporation, Armonk, New York, and a local memory 110. The local memory includes one or more queues 111 used for messaging. In one example, the one or more queues are transmission queues, which include messages to be transmitted to receiver 104.

[0030] Receiver 104 includes, for instance, an operating system 112, such as the TPF Operating System, and one or more destination queues 114 for receiving messages transmitted from sender 102.

[0031] Channel 106, which is used in transmitting messages from the sender to the receiver, is referred to as a sender channel, and is based, for instance, on MQSeries, offered by International Business Machines Corporation, Armonk, New York. MQSeries is described in a publication entitled, MQSeries Intercommunication, IBM Publication No. SC33-1872-03 (March 2000), which is hereby incorporated herein by reference in its entirety.

[0032] Further details regarding operating system 108 are described with reference to FIG. 1b. Operating system 108 includes various components used to control aspects of messaging. In one example, these components include an MQManager 120 used in managing the placing of messages on a

queue and the retrieving of messages from a queue; a transaction manager (TM) 122 used in controlling the initiation of commit and/or rollback operations; a resource manager 124 used in controlling the locking of a queue during commit processing; and a log manager 126 used in managing the logging of events and the processing of a recovery log, in accordance with an aspect of the present invention.

[0033] The operating system components work together to manage events associated with various queues of the communications environment, such as the memory resident queues used for transmitting MQSeries messages. One example of such a memory resident queue is described with reference to FIG. 2.

[0034] Referring to FIG. 2, a queue 200 includes one or more messages, assuming that it is not empty. In particular, the queue includes a first pointer 202 to a first message in a chain of one or more messages of the queue, and a last pointer 204 to a last message of the chain. The content of each message is included in one or more system work blocks (SWBs), each of which is, for instance, 1024 bytes in length.

[0035] The definition of the queue and the contents of the queue are written to a checkpoint 206 at predefined time intervals. Additionally, between checkpoints, updates are written to a recovery log 208. The use of the time-

initiated checkpointing and the recovery log provides persistence of the data (e.g., messages) on the queue.

[0036] One embodiment for writing to checkpoint 206 and recovery log 208 is described in a U.S. Patent Application entitled "High-performance Memory Queue", Chen et al., (IBM Docket No. POU920000185US1), Serial No. 09/790,853, filed February 22, 2001, which is hereby incorporated herein by reference in its entirety.

[0037] In accordance with an aspect of the present invention, if a memory resident queue (e.g., Q1 depicted in FIG. 3) is on an inactive processor 300 (e.g., Processor A) and is thus, inaccessible, then another processor 302 (e.g., Processor B) may take over the queue for the inactive processor. In one embodiment, the take over process is part of a reconstruction technique employed to rebuild and switch ownership of one or more memory resident queues of the inactive processor.

[0038] As shown in FIG. 3, the inaccessible queue, Q1, has two messages (M1 and M2) associated therewith. A copy of M1 is located on a checkpoint record 304 for Processor A, and a copy of M2 is located on a recovery log 306 for Processor A. Since the checkpoint for Q1 only has a record of one of the two messages on the queue, and likewise, since the recovery log only shows the other message, then in order to reconstruct Q1, information is needed from both the checkpoint and the log.

[0039] Thus, as shown in FIG. 4, in order for Processor B to rebuild Q1 of Processor A, Processor B reads in Processor A's recovery log, as well as Processor A's checkpoint, which are both stored, for instance, on shared DASD. Processor B then uses this information to rebuild Q1.

[0040] Subsequent to rebuilding Q1, Processor B replaces the checkpointed version of Q1 with a complete rebuilt version of the queue, as shown in FIG. 5. This rebuilt version is located in Processor B's checkpoint 500 and accurately reflects the queue state and contents before Processor A became inactive.

[0041] One embodiment of the logic associated with reconstructing a queue is described with reference to FIGs. 6-9. In particular, FIG. 6 is a general overview of the logic used to perform the reconstruction, and FIGs. 7a-9 provide further details of various steps of FIG. 6.

[0042] The reconstruction logic is performed by one or more processors, such as one or more active processors, that obtain an indication that one or more queues are to be reconstructed (or a step of the reconstruction is to be performed). The indication may be obtained from a command received by a processor, self-determination of a processor, or by any other mechanism. In this embodiment, not all of the steps of the reconstruction need to be performed by the same processor. For instance, a rebuild step can be performed by one processor and a switch step by another. These steps are further described below.

[0043] Referring to FIG. 6, one step of the reconstruction technique includes rebuilding one or more queues of a processor, such as an inactive processor, STEP 600. For example, the contents of each memory resident queue of the inactive processor to be reconstructed are rebuilt, and the inactive processor's checkpoint is updated with the most recent information. This ensures that the queues appear as they did before the processor became inactive. Then, queue ownership is switched from the inactive processor to an active processor in the communications environment, STEP 602. This includes reading the queue into the active processor's memory. Thereafter, the queue is deleted from the inactive processor's checkpoint, and added to the new owning processor's checkpoint, STEP 604. Each of these steps is described in further detail below.

[0044] One embodiment of the logic associated with rebuilding the contents of a memory resident queue is described with reference to FIGs. 7a-7b. Referring to FIG. 7a, initially, the system recovery log is processed to retrieve from the log any data that is needed or desired for rebuilding the queue, STEP 700. In one example, log manager 126 processes the system recovery log in the reverse order that events were written to the log during normal operation. Thus, the recovery process is presented with recovery data in the order of the most recent event first and the oldest event last.

[0045] In one example, the log manager is only interested in the data that is associated with the queue being rebuilt. Thus, the log manager processes the recovery log by selecting an event off of the log, STEP 702 (FIG. 7b), and then, determining whether it is an event that is of interest to the log manager, INQUIRY 704. For example, if it is an MQSeries queue being rebuilt, then the log manager determines whether the event is an MQSeries event.

[0046] In one example, the various MQSeries events that are considered recoverable include: a Checkpoint process begin message and a Checkpoint process complete message; a Put message and a Get message; a sweep and unsweep of a queue; a sender channel batch list of messages; a sender channel batch commit; a sender channel batch rollback; and a receiver channel synch record. Further details regarding these events are included in the following applications, each of which is hereby incorporated herein by reference in its entirety:

[0047] U.S. Patent Application entitled "Performance Of Channels Used In Communicating Between Senders And Receivers", Chen et al., Serial No. 09/790,781, filed on February 22, 2001;

[0048] U.S. Patent Application entitled "High Performance Memory Queue", Chen et al., Serial No. 09/790,853, filed on February 22, 2001; and

[0049] U.S. Patent Application entitled "Managing Memory Resident Queues To Control Resources Of The Systems Using The Queues", Chen et al., (IBM Docket No. POU920000198US1), Serial No. _____, filed _____.

[0050] Should the log manager determine that it is not an event of interest, then a further determination is made as to whether there are more events, INQUIRY 714. If so, then processing continues at STEP 702. If there are no more events, then processing is concluded.

[0051] However, if the log manager determines that it is an event of interest, then the log manager sends the event to an appropriate resource manager (e.g., an MQSeries Resource Manager), STEP 706. The resource manager has the responsibility of determining whether the event is in-doubt, INQUIRY 708. An event is in-doubt, if it is uncertain whether the event is reflected in the checkpoint. As one example, an event is not considered in-doubt, if a message is both added to and removed from a queue in the same checkpoint scope. In other words, if a message is put to a queue and gotten from the queue before the most recent checkpoint process started, then there is no message in the checkpoint, so the message is not considered in-doubt. Similarly, if a message is put to a queue and gotten from the queue after the most recent checkpoint process is completed, then the message is not considered in-doubt.

[0052] Should an event be considered in-doubt, then a further determination is made as to whether the event is the most recent event performed on a particular message, INQUIRY 710. That is, in this embodiment, the recovery process saves only the most recent event performed on a particular message. For example, if a message is put to a queue, and then, retrieved from the queue, the recovery process only saves the fact that the message was removed from the queue. (In other embodiments, other events may also be saved.) If it is the most recent event, then the event is written, by the resource manager, to a transaction look-aside table, STEP 712.

[0053] The transaction look-aside table is a table in local memory used to save in-doubt events. In one example, the table is organized by queue name and includes the minimum amount of data needed to recover a message. The table only stores in-doubt events, in this embodiment, in order to conserve memory resources. However, in another embodiment, it could be used to store other events, as well.

[0054] Subsequent to writing the event to the transaction look-aside table, a determination is made as to whether there are more events on the system recovery log to be processed, INQUIRY 714. This determination is made by, for instance, having the log manager determine whether there are any further records on the log or whether the records of a defined interval have been processed. If there are more events, then processing continues with STEP 702 "SELECT AN EVENT".

[0055] Similarly, if the No path is taken from INQUIRY 708 or INQUIRY 710, then processing continues with INQUIRY 714. Again, if there are more events, then processing continues with STEP 702. Otherwise, processing of the system recovery log is complete, STEP 716.

[0056] Returning to FIG. 7a, subsequent to saving the in-doubt transaction events in the transaction look-aside table, the queue is rebuilt to the status it had before the owning processor became inactive. This includes merging, in memory of the active processor, the look-aside table with the checkpoint for that queue, STEP 720. For increased performance, this step is performed only if the queue had data saved from the system recovery log. (However, in other embodiments, it can be performed regardless of whether data was saved.)

[0057] In order to perform the merge, one or more actions are taken depending on the type of event. Examples of those actions for each of the following transaction events, which may have been saved from the system recovery log, are outlined below:

[0058] * MQPUT of a message: If the message is not already in the checkpoint version of the queue, add it to the queue; else, make sure the message is marked in the checkpoint as available and committed.

[0059] * MQGET of a message: If the message is in the checkpoint version of the queue, remove it; else, do nothing.

[0060] * Sweep of a queue: For each message in the list of messages swept from memory, if the message is in the checkpoint version of the queue, remove it; else, do nothing.

[0061] * Unsweep of a message: If the message is not already in the checkpoint version of the queue, add it to the queue; else, make sure the message is marked in the checkpoint as available and committed.

[0062] * Sender Channel Batchlist of messages: For each message in the list of messages contained in a particular batch (based on logical unit of work ID [LUWID] of the batch), mark each message with the indicated batch LUWID.

[0063] * Sender Channel Batch Commit: If the batch has been successfully sent, remove all messages in the queue that contain the batch LUWID.

[0064] * Sender Channel Batch Rollback: If the batch was not successfully sent, for each message in the queue that has a matching LUWID, clear the LUWID.

[0065] * Receiver Channel Synch Record: Save the most recent synch record for a given receiver channel.

[0066] Subsequent to merging the look-aside table with the checkpoint, the updated queue is written to the checkpoint for the inactive processor, STEP 722. This concludes the processing associated with rebuilding the contents of the memory resident queue.

[0067] Referring once again to FIG. 6, after rebuilding the contents of the memory resident queue, ownership of the queue may be switched to a processor desirous of taking over the queue for the inactive processor, STEP 602. As examples, the processor may be instructed to take over the queue by an operator command or may programmatically determine that it is to take over the queue (e.g., decides to access the queue). The processor to take over the queue may be the same or different from the processor that rebuilt the queue. One embodiment of the logic associated with this ownership switch is described with reference to FIGs. 8a-8b.

[0068] Initially, the checkpoint of the inactive processor is interrogated to locate the queue whose ownership is being taken over by the active processor, STEP 800. The queue is then read into memory of the new owning processor and given a temporary system name, STEP 802. Thereafter, a determination is made as to whether a queue with the same name (i.e., same actual name) already exists on the new owning processor, INQUIRY 804. If not, then the

queue is renamed back to its actual name and processing of the switch is complete.

[0069] However, if a queue with the same name already exists, then the messages are moved from the temporary system queue to this already existing queue, STEP 806. In order to move the messages from the temporary queue to the existing queue, commit scopes are used such that messages are not lost or duplicated in the event that an unplanned outage occurs in the middle of the queue take over process. One embodiment of the logic associated with moving messages from the temporary queue to the existing queue is described with reference to FIG. 8b.

[0070] Initially, a transaction is begun, STEP 810. Then, a message is retrieved from the temporary system queue, STEP 812, and put on the new queue, STEP 814. Thereafter, a determination is made as to whether a defined number of messages (e.g., X, where X is selected as a comfortable number of messages in one transaction) has been moved, INQUIRY 816. If the defined number of messages has not been moved, then processing continues with STEP 812. Otherwise, the transaction is committed, STEP 818.

[0071] Thereafter, a determination is made as to whether there are more messages on the temporary system queue to be moved to the existing queue, INQUIRY 820. If there are more messages to be moved, then processing continues at STEP 810. Otherwise, processing of the move, and thus, the ownership switch, is complete, STEP 822.

[0072] Referring once again to FIG. 6, subsequent to switching ownership of the queue, the queue is deleted from the inactive processor's checkpoint and added to the new processor's checkpoint, STEP 604. One embodiment of the logic associated with reflecting this change is described with reference to FIG. 9. Initially, the new version of the queue is written from memory of the new owning processor to the new processor's checkpoint stored, for instance, on DASD, STEP 900. Then, the inactive processor's checkpoint is updated, which includes, for instance, deleting the queue from that checkpoint, STEP 902. This concludes one embodiment of a technique for reconstructing a queue.

[0073] In one example, the above procedure is used for each queue to be reconstructed. It may be that all the steps are performed for each queue prior to moving on to the next queue, or that one or more steps may be performed on a plurality of queues before proceeding to the next step. For instance, a plurality of the queues of an inactive processor to be reconstructed may be rebuilt before proceeding to the take over step. Other variations are also possible.

[0074] Although the examples described above describe reconstructing MQSeries queues, such as MQSeries memory resident transmission queues, aspects of the invention can also be used for other memory resident queues, including for queues other than MQSeries queues. Aspects of the invention are applicable for systems other than those using MQSeries.

[0075] Further, although aspects of the invention are described with reference to queues of an inactive processor, aspects of the invention are equally applicable to queues that are inaccessible for other reasons and/or for queues that are to be moved from one processor to another regardless of the reason.

[0076] The communications environment described above is only one example. For instance, although the operating system is described as TPF, this is only one example. Various other operating systems can be used. Further, the operating systems in the different computing environments can be heterogeneous. The invention works with different platforms. Additionally, the invention is usable by other types of environments.

[0077] The present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, computer usable media. The media has embodied therein, for instance, computer readable program code means for providing and facilitating the capabilities of the present invention. The article of manufacture can be included as a part of a computer system or sold separately.

[0078] Additionally, at least one program storage device readable by a machine, tangibly embodying at least one program of instructions executable by the machine to perform the capabilities of the present invention can be provided.

[0079] The flow diagrams depicted herein are just examples. There may be many variations to these diagrams or the steps (or operations) described therein without departing from the spirit of the invention. For instance, the steps may be performed in a differing order, or steps may be added, deleted or modified. All of these variations are considered a part of the claimed invention.

[0080] Although preferred embodiments have been depicted and described in detail herein, it will be apparent to those skilled in the relevant art that various modifications, additions, substitutions and the like can be made without departing from the spirit of the invention and these are therefore considered to be within the scope of the invention as defined in the following claims.